OAK RIDGE National Laboratory

Performance portability experiments with the Wilson-Dslash Kernel from Lattice QCD, using Data-Parallel C++/SYCL and Kokkos

Balint Joo joob@ornl.gov

+ very many co-authors (next page)

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Dramatis Personae

- This work is a continuation of work that many have contributed to. My thanks and acknowledgements to my Co-Authors for this talk below
- NERSC: Jack Deslippe, Thorsten Kurth (now NVIDIA), Doug Doerfler
- Kokkos Team:
  - Sandia National Labs: Christian R. Trott, Dan Ibanez, Dan Sunderland
  - ORNL: Damien Lebrun-Grandie
  - ALCF: Nevin Liber
- NVIDIA: Kate Clark
- Intel: Jeongnim Kim, Patrick Steinbrecher
- AMD: Nick Curtis, Rene van Oostrum

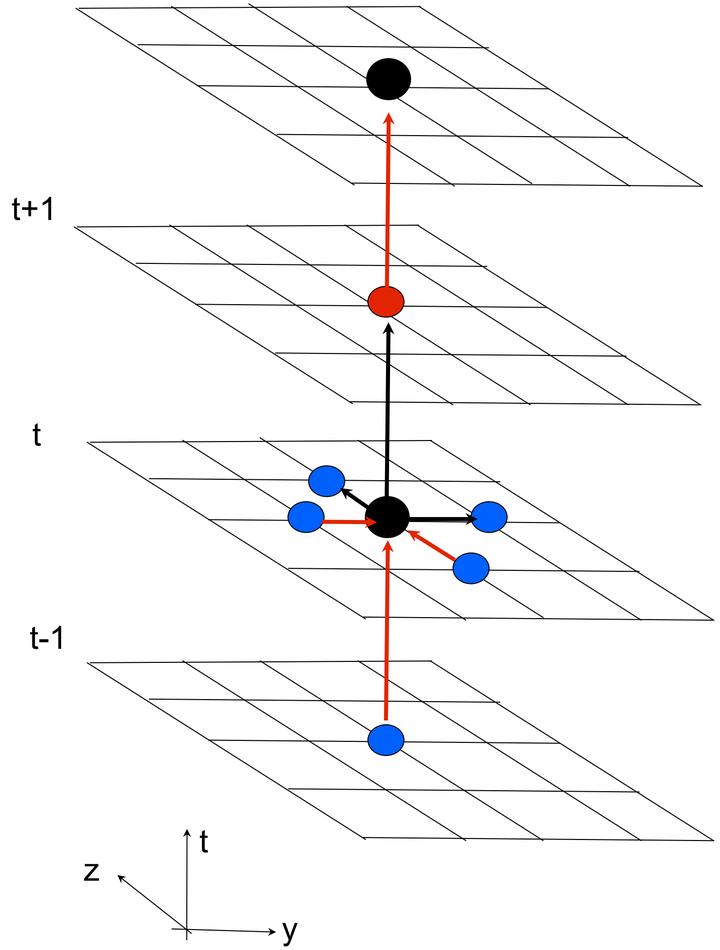
Introduction/Recap from SC'19 P3HPC Workshop

- Lattice QCD codes need to be performance portable for the exascale
- Next set of pre-exascale and exascale systems all use different accelerators
  - NERSC Perlmutter (NVIDIA GPUs)
  - ALCF Aurora (Intel Xe Accelerators)
  - OLCF Frontier (AMD GPUs)
- Many portable programming models available
  - OpenMP, OpenACC, Kokkos, Raja, DPC++/SYCL, HIP, pSTL, …
- Need a "proof of non-death" to evaluate programing models for Lattice QCD codes
- We chose to implement a Wilson Dslash operator as a Mini-App
  - Our choice was to look at Kokkos and DPC++/SYCL

Wilson Dslash Operator in a nutshell

$$D_{x,y} = \sum_{\mu} \left[(1-\gamma_\mu) U_{x,\mu} \delta_{x+\hat{\mu},y} + (1+\gamma_\mu) U^{\dagger}_{x-\hat{\mu},\mu} \delta_{x-\hat{\mu},y} \right]$$

- 4D nearest neighbor, covariant derivative operator
- 3x4 component complex ‘spinors’ on the lattice sites
- 3x3 complex matrices on the links
- (1 +/- γ_μ) are ‘spin projection’ operators
- Ideally can reuse 7 out of 8 neighbors from cache
- due to even-odd coloring: no reuse of links
- AI between 0.87-1.72 depending on neighbor reuse & RFOs
  - Memory Bandwidth Bound on current accelerator architectures for reasonably large lattice



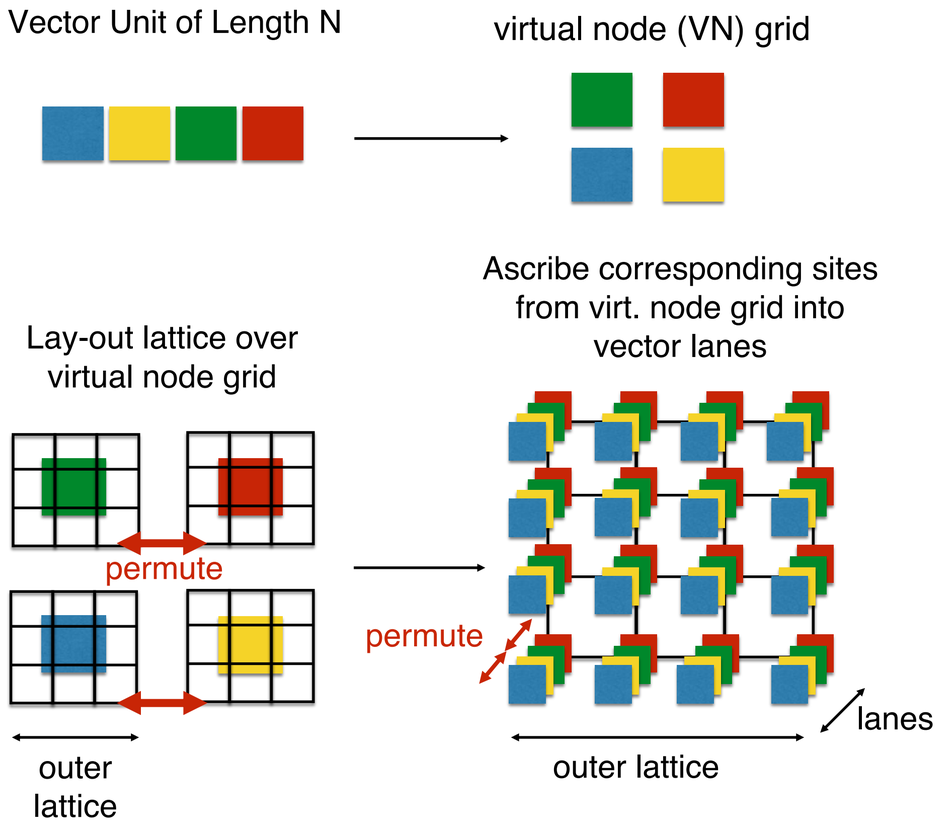
OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

SIMD Vectorization

- Treat SIMD lanes as virtual node (VN) grid
- Layout (block-domains) lattice on VN grid
- Local lattice per lane -> outer lattice
- Nearest-Neighbor access on local lattice boundary = access of neighboring lane's data
  - SIMD shuffle/permutation
  - for $2^{\log N}$ VN grid (up to 16 lanes for 4D) can get neighbor's lane-ID using bitwise XOR
  - for more than 16 lanes, split into power of 2 sets of up to16, and different bitwise XORs needed for each set => use general permutation
- SIMD offers memory access benefits too
  - aligned/coalesced loads & stores

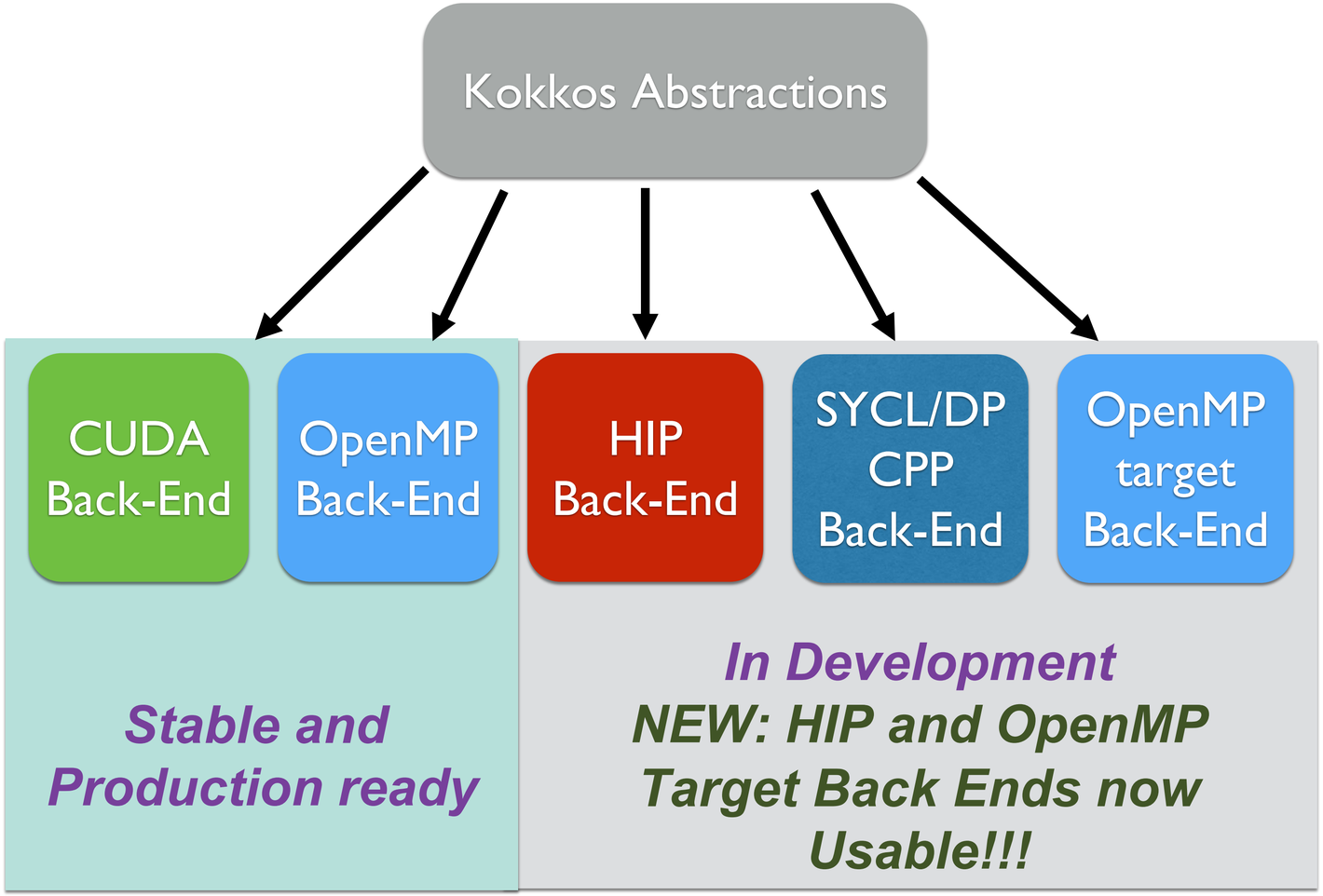
Virtual Node Vectorization (P. Boyle, e.g. in Grid, BFM)

e.g. arXiv:1512.03487[hep-lat]

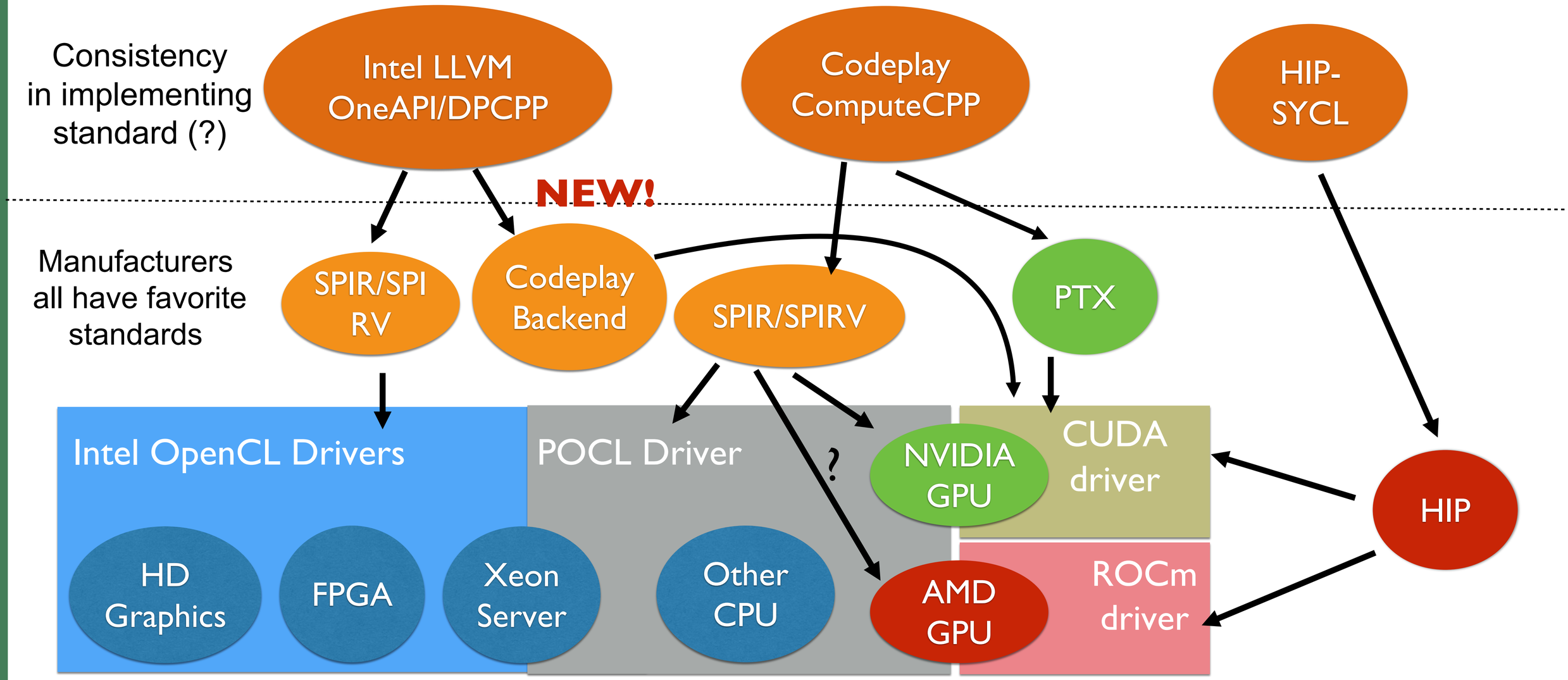


Performance Portability via Kokkos

- Kokkos provides portability via back-ends: e.g. OpenMP, CUDA, …
- Most abstractions are provided in a C++ Header library
  - parallel_for, reduction, scans & more
  - our work uses parallel_for
- Kokkos provides the Kokkos View data-type
  - user can customize index order
  - explicit memory movement only
  - select memory space via policy
- Bind Execution to Execution Space
  - select back end via policy

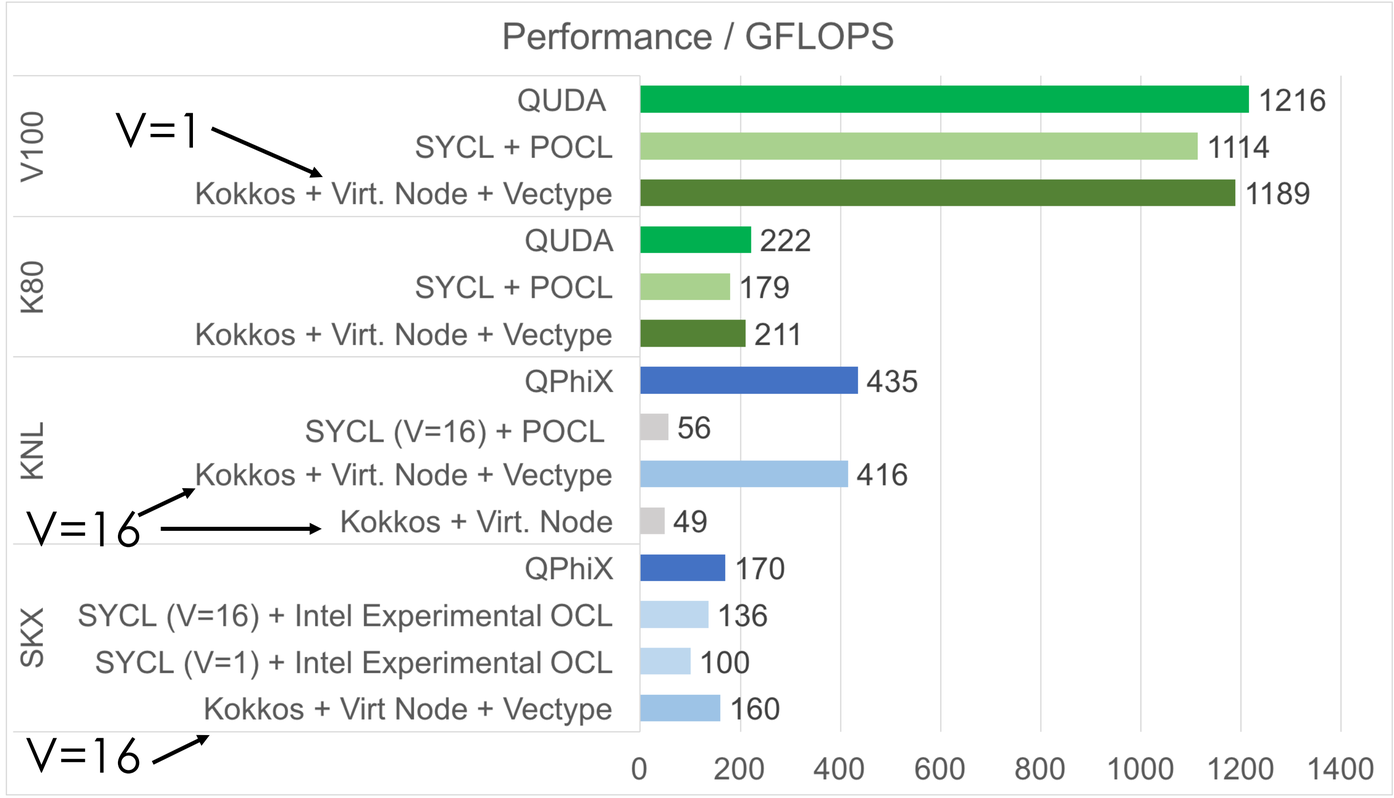


Portability via SYCL



Summary at the time of SC'19 P3HPC workshop

- Kokkos did well on all platforms.
  - CPU targets need a custom SIMD type
  - Hand coded for AVX512
- SYCL/DPC++ did reasonably well
  - using Intel OpenCL runtime on SKX/Gen-9
  - using Codeplay Compiler with POCL OpenCL on V100



What is new

- Tested DPC++ Subgroup Extensions for SIMD
- Tested Codeplay CUDA Back End for Intel Public LLVM compiler
- Tested Kokkos HIP-backend
- Tested Kokkos OpenMP Offload
  - Intel OneAPI ICPX implementation for Gen-9
  - LLVM Clang-10 for NVIDIA-NVPTX back-end
  - Cray-CCE v. 10.0.1 on NVIDIA back end

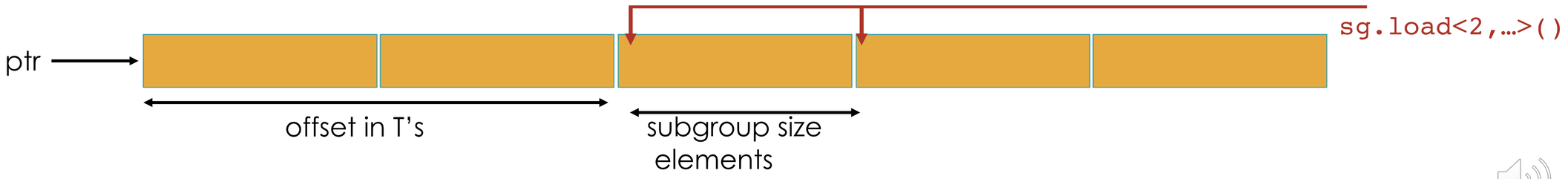
Intel Subgroup Extensions (proposed)

- Similar to Warp/Wavefront level SIMD in CUDA/HIP
- Can get subgroup from nd_item<> index of work-item

```
[[cl::intel_reqd_sub_group_size(VN::VecLen)]]
inline
void operator()(cl::sycl::nd_item<2> nd_idx) const {
   // VN Grid site id.
   size_t site = nd_idx.get_global_id(0);

   // Get the subgroup that I am a part of
   sycl::intel::sub_group sg = nd_idx.get_sub_group();
```

- Single Load/Store per subgroup

```
template<typename T, sycl::access::address_space Space>
inline  MGComplex< typename BaseType<T>::Type>
Load(size_t offset, const sycl::multi_ptr<T,Space> ptr, const sycl::intel::sub_group& sg)
{
   using TypeInMGComplex = typename BaseType<T>::Type;
   sycl::vec<TypeInMGComplex,2> load_vec = sg.load<2,TypeInMGComplex,Space>(ptr + offset);
   return MGComplex<TypeInMGComplex>(load_vec.s0(),load_vec.s1());
}
```



OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Sub Group Permutations

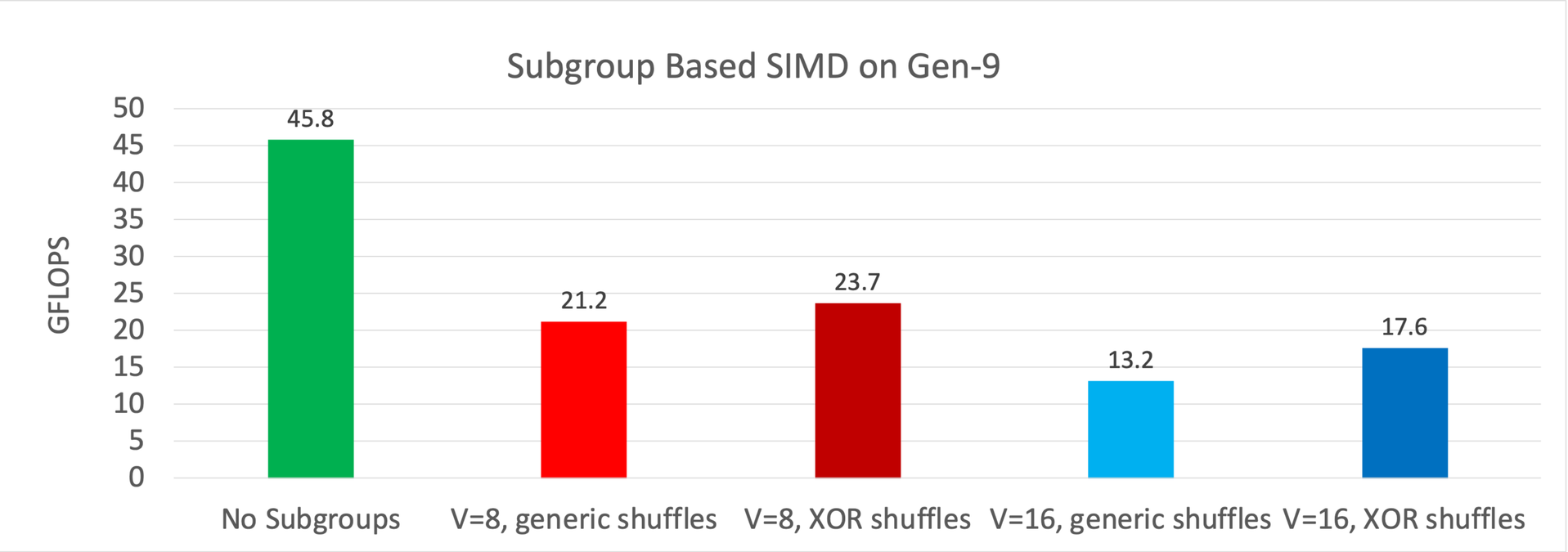
Bitwise XOR Based

```
template<> inline
MGComplex<float> Permute<float,8>::permute_xor_Y( const MGComplex<float>& in,
                                                  const sycl::intel::sub_group& sg) {
  return MGComplex<float>{ sg.shuffle_xor(in.real(), {1}),
                           sg.shuffle_xor(in.imag(), {1}) } ;   // XOR mask is 0xb001, {} to init id<1>
```

Generic

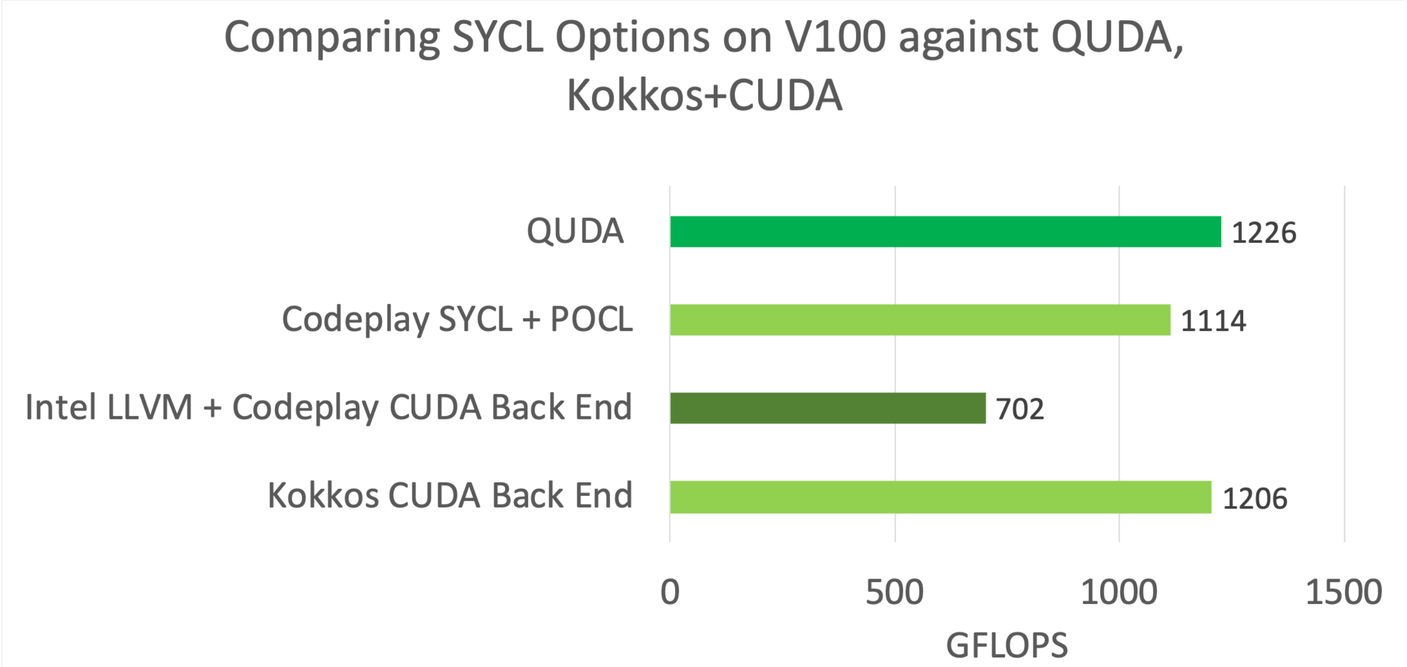
```
// Masks for SUBGROUP SIMD
static constexpr std::array<int,8> x_mask = {0,1,2,3,4,5,6,7};  // Mask array for element 'i'
static constexpr std::array<int,8> y_mask = {1,0,3,2,5,4,7,6};  // stores source lane for lane 'i'
static constexpr std::array<int,8> z_mask = {2,3,0,1,6,7,4,5};  //
static constexpr std::array<int,8> t_mask = {4,5,6,7,0,1,2,3};  //

template<typename T, int N>
static inline MGComplex<T> permute(const std::array<int,N> mask,
                                   const MGComplex<T>& in,
                                   const sycl::intel::sub_group& sg)
{
  MGComplex<T> ret_val;
  ret_val.real( sg.shuffle( in.real(), mask[ sg.get_local_id()[0] ] ) );  // Look up source_id in mask[]
  ret_val.imag( sg.shuffle( in.imag(), mask[ sg.get_local_id()[0] ] ) );  // sg.get_local_id() return 'lane-id'
  return ret_val;
}
```

Preliminary Subgroup SIMD Results



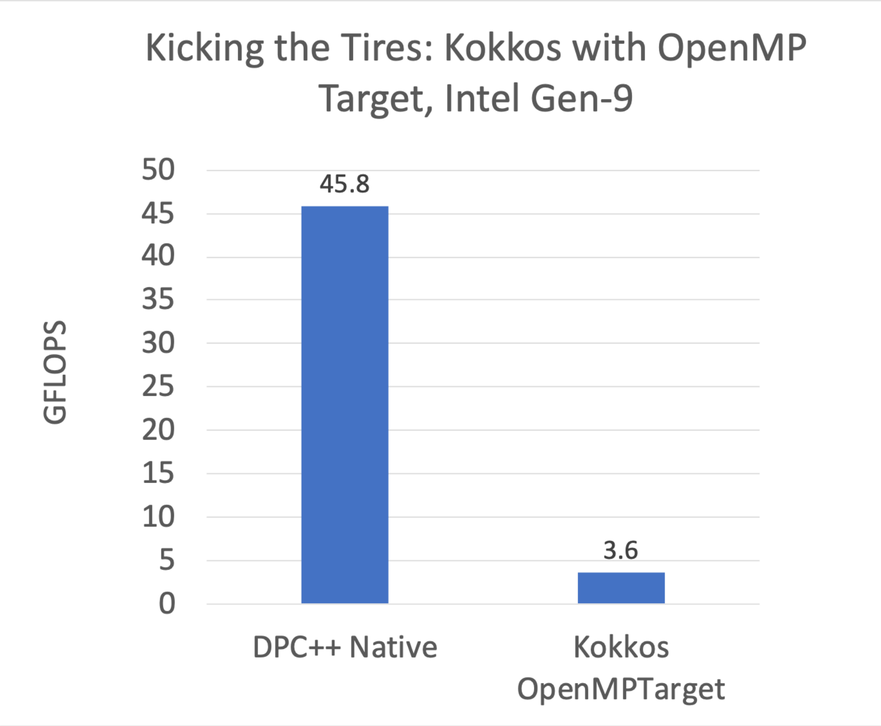
- Tests used lattice with 32^4 sites, Intel OneAPI public beta=8 compiler (clang), Intel Iris Pro Graphics 580 with 128MB EDRAM
- In these tests using subgroup based SIMD did not provide a benefit on Gen-9, best performance is naive V=1 case.
- Further investigation could be interesting as well as test on a Xeon part.

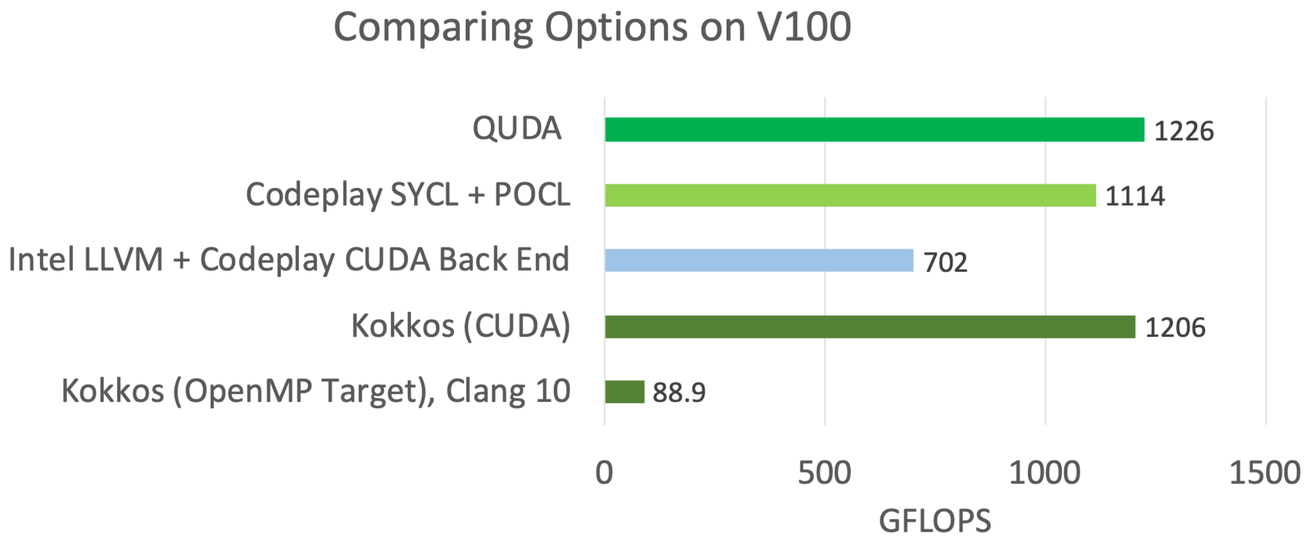
DPC++ with CUDA Back End from Codeplay



- Lattice with 32^4 sites, dpc++/11.0.0-20200721 from Doug Doerfler on Cori GPU based on LLVM11
- Compiler built the plain SYCL code and the code ran fine
- Performance is not as good as Codeplay SYCL + POCL
- Compiler compiled code using subgroup extensions but had linking issues
  - not really a surprise as I didn't expect support for extensions

Kokkos and OpenMP Target





- Good news: OpenMP Target for Kokkos is functional in several compilers
- Bad News: Currently, Out of the Box performance seems poor — may need specific optimization/tuning

Kokkos with HIP Backend

- ORNL is working with AMD on a HIP Backend for Kokkos (led by Damien Lebrun-Grandie)
- WilsonDslash mini-app over Kokkos compiles and runs with the HIP back end
- Performance was low, compared to expectations based on available memory bandwidth (1 TB/s peak mem BW on MI50/MI60 devices)
- We went through a deep dive with Nick Curtis and Rene van Oostrum from AMD
  - found: there seemed to be high register use, and low occupancy and some amount of register spilling going on
  - changing out neighbor index calculations to table lookup made a considerable difference: reduced register count & spilling, increased performance by ~1.24x
  - integer division appears to be expensive on MI50, possible to fix with libdivide ?
  - Moral: cannot escape hardware, the algorithmic choice of computing the index of a neighbor or looking it up may not be performance portable => Allow for choice in implementation

Conclusions & Future Work

- Considerable progress since SC'19 in compilers
  - DPC++ with CUDA Back End now tested and working, Sub Group extension works in native back end.
  - Kokkos tested with OpenMP-Target backend using several compilers
  - Kokkos HIP backend is quite functional from point of view of Dslash Mini-App
- Performance is not uniformly good yet accross the board
  - e.g. did not expect DPC++ with Subgroup SIMD to be outperformed by non SIMD code
  - OpenMP Target performance may need to be tuned/investigated
- Future work: Kokkos has a public SIMD Type in https://github.com/kokkos/simd-math
  - I have added a proposal for permutes in a private branch (PR#18)
  - Still to integrate these into KokkosDslash
- Lessons learned:
  - Fantastic that so many OpenMP implementations now compile and run
  - Cannot escape hardware features (e.g. integer divisions etc).

Acknowledgements

- B. Joo acknowledges funding from the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the Exascale Computing Project (2.2.1.01 ADSE03 Lattice QCD)
- B. Joo acknowledges funding from the U.S. Department of Energy, Office of Science, Offices of Nuclear Physics, High Energy Physics and Advanced Scientific Computing Research under the SciDAC-4 program.
- B. Joo acknowledges travel funding from NERSC for a summer Affiliate Appointment for work on Kokkos
- The 2017 ORNL Hackathon at NASA was a collaboration between and used resources of both the National Aeronautics and Space Administration and the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory. Oak Ridge Nation Laboratory is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- We gratefully acknowledge use of computer time at JeffersonLab (SciPhi XVI cluster), K80 Development node, NERSC Cori and Cori-GPU, OLCF Summit, Lyra, and the Cray Frontier Center of Excellence Tulip System